Coordinates, Conversions, and Kinematics for the Rochester Robotics Lab

Christopher M. Brown Raymond D. Rimey

Technical Report 259
August 1988

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

-f

Coordinates, Conversions, and Kinematics for the Rochester Robotics Lab

Christopher M. Brown and Raymond D. Rimey

TR 259

August 1988

Abstract

This is a guide to coordinate systems, representations, and geometric relationships between them, for components of the Rochester Robotics Laboratory. The main entities at issue are the joint angles, location variables, and coordinate systems of the Puma, the camera angles and coordinate systems associated with the head, the spatial location of three-dimensional points, and the kinematic and inverse kinematic relationships between them. The robot-to-camera kinematic chain is described, conversions between homogeneous transformations and VAL location descriptions are provided, and inverse problems (camera angles to aim cameras at a 3-D point given a robot configuration, binocular stereo calculations) are solved. Constants describing the robot head and sample robot description data structures are provided.

(KF)

This work was supported by the DARPA U.S. Army Engineering Topographic Labs Grant No. DACA76-85-C-0001, and by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, NY, 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC, 20332, under Contract No. F30602-85-C-0008. This contract supports the Northeast AI Consortium (NAIC). The Xerox University Grants Program provided equipment used in the preparation of this paper. Peggy Meeker and Rose Peet produced the graphics.

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 259	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Coordinates, Conversions, and Kinematics for the Rochester Robotics Lab		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Christopher M. Brown Raymond D. Rimey		8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Computer Science 734 Computer Studies Bldg. Univ. of Rochester, Rochester, NY 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA / 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE August 1988 13. NUMBER OF PAGES 19
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Puma Robot
Puma arm
Rochester "head"
coordinate systems
coordinate conversions

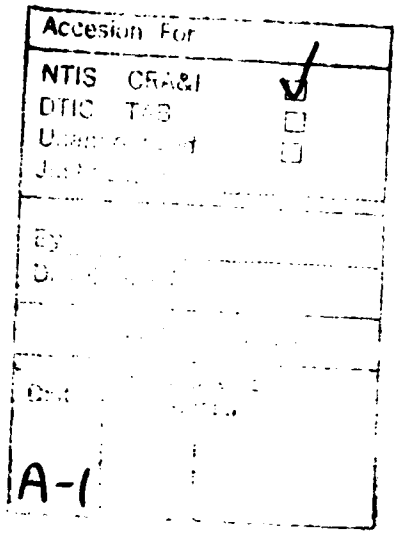
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This is a guide to coordinate systems, representations, and geometric relationships between them, for components of the Rochester Robotics Laboratory. The main entities at issue are the joint angles, location variables, and coordinate systems of the Puma, the camera angles and coordinate systems associated with the head, the spatial location of three-dimensional points, and the kinematic and inverse kinematic relationships between them. The robot-to-camera kinematic chain is described, conversions between homogeneous transformations and VAL location descriptions are provided, and inverse problems (camera angles

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

20. ABSTRACT (Continued)

to aim cameras at a 3-D point given a robot configuration, binocular stereo calculations) are solved. Constants describing the robot head and sample robot description data structures are provided.



1. Purpose

The purpose of this document is to relate elementary kinematic concepts and calculations to the University of Rochester Robotics Laboratory, with the aim of making certain aspects of the Puma arm and the two-camera head easier to use and understand. In using the robot to interact with the world, one quickly finds a potentially bewildering set of coordinate systems, angles, parameters, and state descriptions that must be related one to another in order to produce coherent robot actions and to answer common robotic questions. This document mainly concerns the definition and manipulation of coordinate systems. The semantics of the coordinates are such things as tool positions, camera orientations, and so forth.

Section two presents a short section on transformation notatation and properties, which should be read. There follows a glossary of scalars, vectors, and transformations we use later in the document, which can be skimmed and referred to as needed.

Section 3 defines some important robotic coordinate systems. LAB is the base coordinate system attached to the laboratory. TOOL describes the location of the robot head. FLANGE is another description for head location, but one more convenient for use with imaging operations.

Section 4 describes the model of the imaging process. Section 5 describes and defines the transformations along the chain of links that the Puma and the head embody.

There follow several sections, each one describing how to convert from one representation to another, or deriving a desired transformation or description from a specification. For example, it may be of interest to convert the TOOL coordinate system into the description used by VAL for robot location. An example of deriving an interesting configuration is to find the camera altitude and azimuth angles that the camera at a point in (X,Y,Z) space.

There is room for expansion of this report, and an expanded version should be produced later when more is known. One obvious lack at present is the Jacobian calculations for the head -- at what rate to move the cameras to compensate continuously for continuous head motion and vice-versa.

2. Transform Basics

We follow [Paul 1981] and represent 3-space points as column homogeneous 4-vectors or column Cartesian 3-vectors. Transforms (and coordinate systems, or CS's), are homogeneous 4x4 matrices. All transforms but the camera transform are rigid. Thus they denote a rigid rotation or translation or both. If both, then think of the rotation as being done before the translation. A transform B operates on points expressed as column vectors to yield new points. A transform B represents a CS in that it can be thought of as four columns, three of which represent points at infinity and correspond to directions of the X,Y, and Z axes of a CS, and the last of which represents a 3-space point and corresponds to the origin of the CS. Transforming (that is, multiplying) a CS by a transform just rigidly moves the CS in space. LAB is the identity transform, and transforming LAB by B yields B. Thus B cleverly represents a coordinate system and a transform that moves LAB to that coordinate system.

If $\vec{x}$ is a vector denoting a point in LAB coordinates, and A and B are transforms, then $B\vec{x}$ gives the coordinates of ("is") the point, in LAB coordinates, that results from

rotating and translating $\vec{x}$ by B. AB$\vec{x}$ is the point resulting from applying B to x, then A to the result, where rotating and translating by A is done with respect to the original LAB coordinate system. Alternatively, AB$\vec{x}$ means applying to $\vec{x}$ the transformation A, followed by the transform B *expressed in the frame* A. is conceptualized as taking place in the coordinate system induced by all previous movements, the final transform is $A_1 A_2 \cdots A_n$ if 1 is the link connected to LAB. If $\vec{x}$ is a point in LAB and B is a frame, then $\vec{x}$ expressed in B is $B^{-1}\vec{x}$. Last, if B takes LAB to the CS B (its alibi aspect), then B also is the transform from B coordinates to LAB coordinates (its alias aspect).

2.1. Scalars

VAL expresses distance in mm, angles in degrees. Parameters to our subroutines are expressed in radians, to save conversions. Since users often prefer degrees, our user interfaces (and this document) usually express angles in degrees.

f	Effective imaging system focal length (including digitizing effects). (Note that this number has nothing to do with the physical focal length of the lens).
s	Camera aspect ratio: column spacing / row spacing.
ϕ	Camera platform altitude angle in radians.
θ	Generic azimuth angle.
θ_L	(Head's) Left Camera azimuth angle.
θ_R	Right Camera azimuth angle.

2.2. Vectors

Pixel coordinates in MaxVideo routines are expressed as (pixel offset in scanline, scanline), which corresponds to the "physical" (x,y) coordinate scheme used in the imaging model coordinate system, which has Y axis down and X axis to the right. In array indexing, however, the "natural" element-addressing scheme is Image[row][column], which has semantics (y,x). Where this document uses pixel coordinates, they are expressed in the physical system rather than in the array-indexing system.

$\vec{x}$	generic vector $(x,y,z)^T$, a point in space expressed in some coordinate system. Often the homogeneous column 4-vector $(x,y,z,w)^T$
$(\hat{x},\hat{y})$	image (pixel) coordinates of a point.
(O,A,T)	"Orientation, Altitude, and Twist" angles describing the orientation of TOOL axes in terms of LAB. Like Euler angles but not (see below).
Loc	(X,Y,Z,O,A,T). A generic location (X,Y,Z position and orientation) used by VAL. May be relative. In this document we usually construe Loc to define the location of the TOOL CS in terms of LAB.
Head	(ϕ, θ_L, θ_R), Camera rotations defining the head configuration.
JntAngs	six angles defining the rotations of the robot links. Used by VAL software as "precision points".

2.3. Matrices, Coordinate Systems, and Transforms

S_{ij}	The i^{th} row, j^{th} column element of S.
A_i	An "A Matrix", expressing the rigid transform induced by one link in a kinematic chain.
T_j	A transform induced by a kinematic chain. By definition, $T_j = A_1 A_2 \cdots A_j$.
Rot_x(a)	Rotation around X axis by angle a.
Rot_y(a)	Similar
Rot_z(a)	Similar
Trans(x,y,z)	Translate by x,y,z.
LAB	CS attached to the laboratory, defined below.
TOOL	A user-definable coordinate system rigidly attached to joint 6 of the Puma.
NULLTOOL	VAL's default value for TOOL. It corresponds to a relative location of (X,Y,Z,O,A,T) = (0,0,0,90,-90,0).
FLANGE	CS convenient for head and camera calculations. Like another TOOL CS, it is rigidly attached to joint 6. Defined in terms of NULLTOOL or T_6 -- relative to T_6 it has a Location of (0,0,0,-180,0,-90) (see below).
C	A perspective camera transform, not a rigid CS transform.
CamPos	FLANGE transformed so its Z axis points along a camera's optic axis and its origin is at the front principal point of the lens. CamPosL and CamPosR are for left and right cameras.
PhysPixel	Translates pixel coordinates so origin is upper left corner of pixel array, not its middle.

3. Coordinate Systems and Robot Coordinates

3.1. LAB

The Puma's internal representations assume that its first link is rigidly attached to a LAB coordinate system. VAL generally reports locations in LAB coordinates. The Puma's BASE coordinate system is usually synomynous with LAB. BASE may be changed by invoking the VAL BASE command, which allows translation of the X,Y,Z, origin and Z-rotation of BASE. An automatic Z rotation may be a good idea, since the Puma is bolted slightly askew.

At initialization, according to the manual, the origin of LAB is at the intersection of Joints 1 and 2. Certainly the origin is somewhere near the centerline of the Puma. Imagine you are looking in at the Puma through the window. Then in the default state, the LAB X axis points to your right parallel to the window, the Y axis is pointing away from you toward the far wall, and Z is up (Fig. 1). If all joint angles are 0 (JntAng = $\vec{0}$), TOOL = NULLTOOL, and Head = (0,0,0) = $\vec{0}$, the cameras are pointing away from you down TOOL (and LAB) Y.

3.2. T_6 and TOOL

T_6 is a CS attached to the end of the last link of the robot. The TOOL CS is defined relative to T_6. T_6, TOOL, and FLANGE share a common origin. T_6 is only useful to understand the TOOL coordinate system. When JntAng = $\vec{0}$ (Fig. 1), T_6 has its X axis pointing down (along LAB -Z), its Y axis pointing along LAB X, and its Z axis along LAB -Y (Fig. 1).

The TOOL coordinate system is a transformation of T_6, and is a primitive notion in VAL commands, which can often be expressed in TOOL coordinates. Upon initialization, TOOL = NULLTOOL, which (Fig. 1) makes T_7 simply a translation of LAB if JntAng = $\vec{0}$. VAL reports the location of the TOOL CS in the form Loc = (X,Y,Z,O,A,T). NULLTOOL corresponds to a relative location, with respect to T_6, of

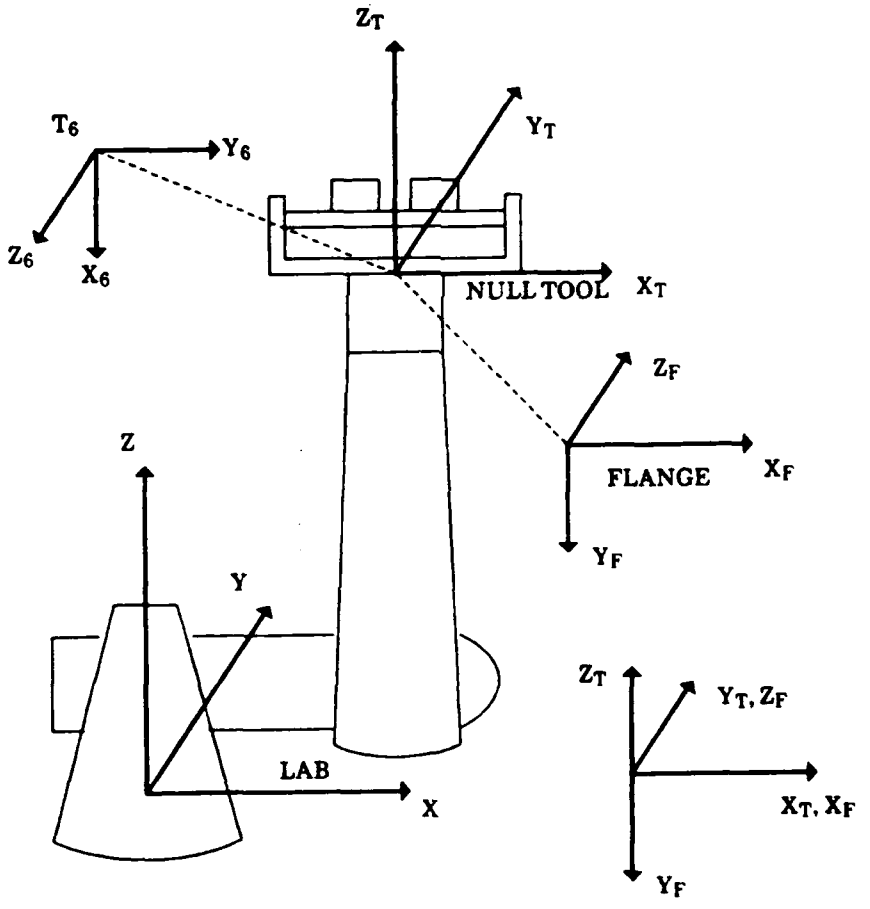


Figure 1: The Puma and head, as seen from the observation window, showing three basic CSs. The origins of T_6, NULLTOOL and FLANGE coincide. The robot is shown with JntAng = $\vec{0}$ and Head = $\vec{0}$. In this configuration Loc = (650,190,975,90,-90,0).

(0,0,0,90, -90, 0). The O,A,T components of Loc are angles that have the following semantics. "Rotate by -O around (the current) X, then by A around the new Y, then by T around the new Z". Thus they have the same flavor as Euler angles. It is easy to verify that applying the NULLTOOL transform to T_6 at JntAng = $\vec{0}$ transforms T_6 to have axes parallel to LAB.

One interesting and useful aspect of TOOL is that it can be redefined by the user. For instance, one can redefine tool as a remote point, such as a world point that is currently in view. Then it is possible to issue VAL commands that rotate the robot head around the TOOL origin. The effect is for the head to move in space and to be continuously reoriented by the robot wrist (not the camera motors) so that the cameras remain pointed at the same three-dimensional scene point.

3.3. FLANGE

The head is rigidly attached to the sixth robot link, and hence to T_6 and TOOL. When the eyes are facing "forward" (Head = $\vec{0}$), FLANGE is a coordinate system whose axes are oriented to be consistent with the camera imaging model (Fig. 1). In FLANGE, Z is out along the direction the head is facing (parallel to the optic axis of cameras if Head = $\vec{0}$). Y is down, increasing with the row number addresses of pixels in an image, and X is "to the right", increasing with the column numbers of pixel addresses in the image. One common trick is to define TOOL as FLANGE. This renders the explicit FLANGE transform unnecessary. If the TOOL transform is set to (X,Y,Z,O,A,T) = (0,0,0,-180, 0, -90), then T_6 is transformed to FLANGE by the TOOL transform within the Puma.

4. Camera Imaging Model

Here we are concerned with the "intrinsic" camera parameters [Tsai 1986], which govern its optical properties. "Extrinsic" properties define its location in space, and determine the CamPos coordinate system. These properties are determined by the kinematic issues discussed below. For intrinsic camera properties we use a pinhole model (e.g. [Duda and Hart 1973]), which is to say we do not correct for radial lens distortions. This is not a policy, it is just that we have not yet been motivated to do so.

The camera optic axis is out along the positive Z axis. Looking out along the camera's line of sight, Y points down, increasing as does the scan-line number in the camera's image. X point to the right, increasing as the pixel number along a scan line. X,Y,Z form a right-handed coordinate system. We assume the origin of coordinates is at the camera's front principal point, and that the image is formed at a distance f in front of the origin in the X-Y plane by point projection. Then a scene point $\vec{x}$ yields the image point coordinates

$$(\hat{x},\hat{y})=(\frac{xf}{z},\frac{ysf}{z}),$$

where f is the effective focal length of the entire imaging, transmission, digitization, and ROIStoring process, and s is a scaling constant that expresses the "aspect ratio" of the system. The angular (spatial) resolution of the final pixels resting in ROI is less in the Y direction, and s tells by how much. Thus the model tells where a point appears (under default setings) in ROI store, and under default settings where its location is reported by

FeatureMax. It includes all effects induced by CCD chip layout, conversion to analog waveform by the Panasonic electronics, sampling by DigiMax, and storage in ROI. It does not pretend to say anything about any of these effects in isolation.

The parameters f and s were estimated using a calibration chart, and the values in the "Constants" section represent our best current estimates.

The camera transform can be expressed as multiplying the homogeneous scene point vector by a transform matrix C and then performing normalization [Tsai 86]. The normalization operation scales a homogeneous 4-vector $(x,y,z,w)^T$ by (1/w). In this context, the resulting value of z is an artifact, since the image has only two dimensions. The matrix C is

$$C = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & fs & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The camera extrinsic properties are determined by the LAB-Camera kinematic chain discussed next.

5. The LAB-Camera Kinematic Chain

There are some fourteen identifiable transforms between LAB and a CamPos Coordinate System (Table 1). The head transforms can be collapsed into two link transforms involving offsets and one rotation each [Paul 81], but in this treatment all the transforms beyond A_8 are pure rotations or translations. The Joint 1-6 transforms A_1–A_6 generally involve both offsets and rotations.

Define T_i as $A_1 A_2 \cdots A_i$. These transforms, their partial products, and the inverses of their partial products, are of use in everyday robotic life. For instance, T_8 converts points expressed in FLANGE coordinates (often the output of vision routines is in FLANGE) into LAB. As another example, to simulate making an image with a camera, a point in LAB must be transformed by T_{14}^{-1} in order for the camera imaging model to apply.

Note that T_7 is implemented internally in VAL. We can only ask or set the value of T_7, (not A_1 -- A_7). A_8 -- A_{14} are transforms that are created and manipulated by the user. Thus we can describe A_7 and A_8 as follows.

A_7 NULLTOOL or set by user.

A_8 Rot_x(–90) if TOOL = NULLTOOL, Identity if TOOL = FLANGE.

There are two CamAxis transforms, corresponding to the offsets of left and right cameras along the camera platform. CamAxisL and R are given special names only because they are the last head points that are rigidly affixed to FLANGE. Thus they may offer some efficiency for position evaluations when the eyes are moving but the head (robot) is not. From this point on there are two kinematic chains, corresponding to the differing offsets and azimuth angles of the two cameras, and denoted by L and R. We often group $A_8 \cdots A_{14}$ into a single transform matrix (named CamPosL or CamPosR) expressing the camera location in FLANGE coordinates.

A matrix	Name	Const. or Var.	Resulting CS
Ident	LAB	C	LAB
A_1	Joint 1	V	
A_2	Joint 2	V	
A_3	Joint 3	V	
A_4	Joint 4	V	
A_5	Joint 5	V	
A_6	Joint 6	V	T_6
A_7	Tool	V	TOOL
A_8	FLANGE	C	FLANGE
A_9	Neck Offset	C	
A_{10}	Eye X Offset	CL, CR	CamAxis (L,R)
A_{11}	Altitude	V	
A_{12}	Alt. Offset	C	
A_{13}	Azimuth	VL, VR	
A_{14}	Az. Offset	C	CamPos (L,R)

Table 1: The LAB to camera kinematic chain of transforms.

The definition of distances and angles for the robot head are shown in Fig. 2. See the section on Constants for numeric values.

6. Forward Kinematics: CamPos from (ϕ,θ)

The camera motor control software positions a camera at a specific altitude, or pitch (ϕ) and azimuth, or yaw (θ). We should like to compute CamPos from altitude and azimuth. CamPos is expressed in FLANGE coordinates.

$$CamPos = A_9 A_{10} \cdots A_{14}.$$

The values of the relevant A matrices are as follows (Fig. 2). Constant values are given in Section 13.

A_9 Trans(0,NECK_OFFSET,0).

A_{10} Trans(LEFT_OFFSET,0,0) or Trans(RIGHT_OFFSET,0,0).

A_{11} Rot_x(ϕ).

A_{12} Trans(0,ALT_OFFSET,0).

A_{13} Rot_y(θ_L) or Rot_y(θ_R).

A_{14} Trans(0,0,AZ_OFFSET).

7. Forward Kinematics: TOOL from Loc

VAL provides several useful conversions. A "precision point" is a JntAng vector, and VAL understands robot locations in both precision points and locations (Loc

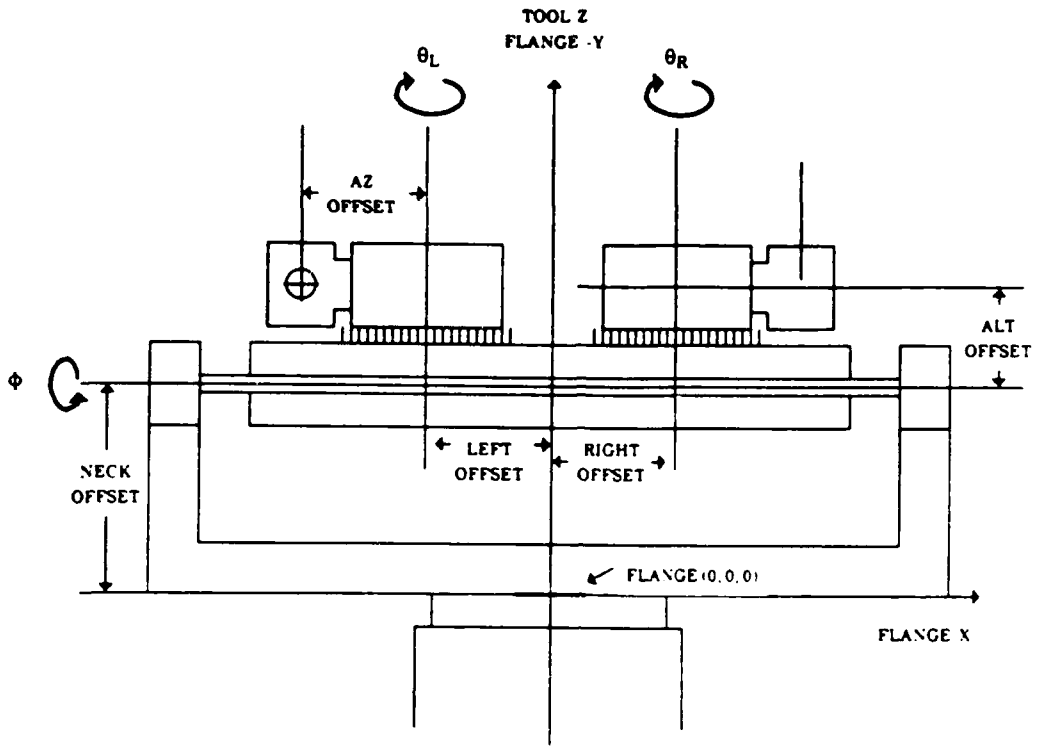


Figure 2: Axes and link offsets in the robot head (see Section 13).

vectors). It of course is possible to derive (O,A,T) simply by composing all the joint angle rotations. Deriving X,Y,Z from the joint angles involves knowing offsets and doing a full forward kinematics solution. Generally then the TOOL location is most easily obtained from VAL. VAL reports the TOOL location as a JntAng vector (a VAL "precision point") and a Loc vector (a VAL "location"). Our current Purdue robot control software only returns a Loc, although that is subject to change.

Converting the (O,A,T) angles to a transform involves knowing exactly what they mean. The Puma manual is not explicit here. Ray Rimey determined the following transformation, which, in its alias aspect, moves LAB to the current TOOL CS.

TOOL(Loc) = Rot_z(-90)Rot_y(90)Rot_x(-O)Rot_y(A)Rot_z(T)Trans(X,Y,Z).

This transformation, in its alibi aspect, thus converts points from TOOL to LAB coordinates. It is written out explicitly below. If the TOOL transform is redefined by the user, then it is to that redefined CS that LAB will be transformed. Redefining TOOL with a VAL command means setting the values of X,Y,Z,O,A,T in the above transformation. Thus if TOOL is redefined within VAL from NULLTOOL to FLANGE,

then A_8 should be the identity transform, and can vanish from the kinematic chain and from the user's external calculations.

8. Inverse Kinematics: (ϕ,θ) from CamPos

Given a camera position expressed as a CamPos transform in FLANGE coordinates, what θ and ϕ angles created it? To find out, write the transform

$$E=A_9 A_{10} \cdots A_{14},$$

and notice that certain individual elements of E contain exactly the sines and cosines of ϕ and θ. We see

$$\phi=atan\_2(E_{21},E_{11}),$$

$$\theta=atan\_2(E_{02},E_{00}).$$

This is a very simple version of the work needed to get O,A,T from TOOL.

9. Inverse Kinematics: O,A,T from TOOL

The FRAME command in VAL takes four input vectors that describe the TOOL axis unit vectors and origin, and returns the corresponding Loc vector (X,Y,Z,O,A,T). The interesting part of this is of course deriving (O,A,T) from the TOOL CS. In turn, this is an operation quite closely related to deriving Euler angles from a transform, which is an early exercise in [Paul 81].

The approach is to multiply the five matrices from the TC..L(Loc) formula (leaving out the translation) together to get a TOOL transform D. Say

$$D=B_1 B_2 B_3 B_4 B_5.$$

Then, postmultiply both sides by B_5^{-1}, and look for interesting relationships elementwise between the two matrices. In this case, as in Paul's solution for Euler angles, we find that we have enough information to compute O,A,T in the form of atan_2() functions, which have good properties. Proceeding to details, use the notation S_A for $sin(A)$, etc, substitute A_i for B_i in the kinematic chain equation above, and write the resulting product of the first four B matrices as

$$B_{1-4}=\begin{bmatrix} -S_O S_A & C_O & S_O C_A & 0 \\ C_O S_A & S_O & -C_O C_A & 0 \\ -C_A & 0 & -S_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The complete transformation is

$$D=\begin{bmatrix} C_O S_T - S_O S_A C_T & C_O C_T + S_O S_A S_T & S_O C_A & 0 \\ S_O S_T + C_O S_A C_T & S_O C_T - C_O S_A S_T & -C_O C_A & 0 \\ -C_A C_T & C_A S_T & -S_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Section 7 establishes that $B_{1-4}=DB_5^{-1}$ = D Rot_z(-T). Performing the formal multiplication on the right hand side yields a 4x4 matrix whose elements are functions of

the elements of D (which we know), C_T, and S_T. Equating these elements to those of B_{1-4}, we find our first interesting equation:

$$S_T D_{20}+C_T D_{21}=0,$$

which implies

$$T=atan\_2(D_{21},D_{20}).$$

We can also read off that

$$-S_A=D_{22}$$

and

$$-C_A=C_T D_{20}-S_T D_{21},$$

so

$$A=atan\_2(-D_{22},S_T D_{21}-C_T D_{20}).$$

Finally, we can read off expressions for C_O and S_O to get

$$O=atan\_2(S_T D_{10}+C_T D_{11},S_T D_{00}+C_T D_{01}).$$

At the time of writing, these derivations have not been checked against the output of the FRAME command.

10. Inverse Kinematics: (ϕ, θ) from (x,y,z)

Given a point $\vec{x}$ at $(x,y,z,1)^T$ in FLANGE, which ϕ and θ parameters will center the point in a camera's view? Following the strategy of the last section did not immediately lead to a promising set of equations. The hope was that the camera physical transform E could be written out and the fact that $E\vec{x}=(0,0,z,1)^T$ would lead to something simple. It did not seem to. Instead we use a straightforward geometric approach (Fig. 3).

From Fig. 3, we have

$$h=(z^2+y^2)^{1/2},$$

$$b=asin(d/h),$$

$$\phi=atan\_2(-y,z)-b.$$

The asin() is bad practice because of its ambiguity and lack of differentiation for angles near 90, but for small angles, as will usually be the case in such a setup as this, it behaves reasonably.

It remains to determine the azimuthal rotation. Rotate space by Rot_x(-ϕ), bringing the cameras and point into a plane of constant y. The point's new (x,z) coordinates become (x, zcos(ϕ)-ysin(ϕ)), and finally we have

$$\theta=atan\_2(x,zcos(\phi)-ysin(\phi)).$$

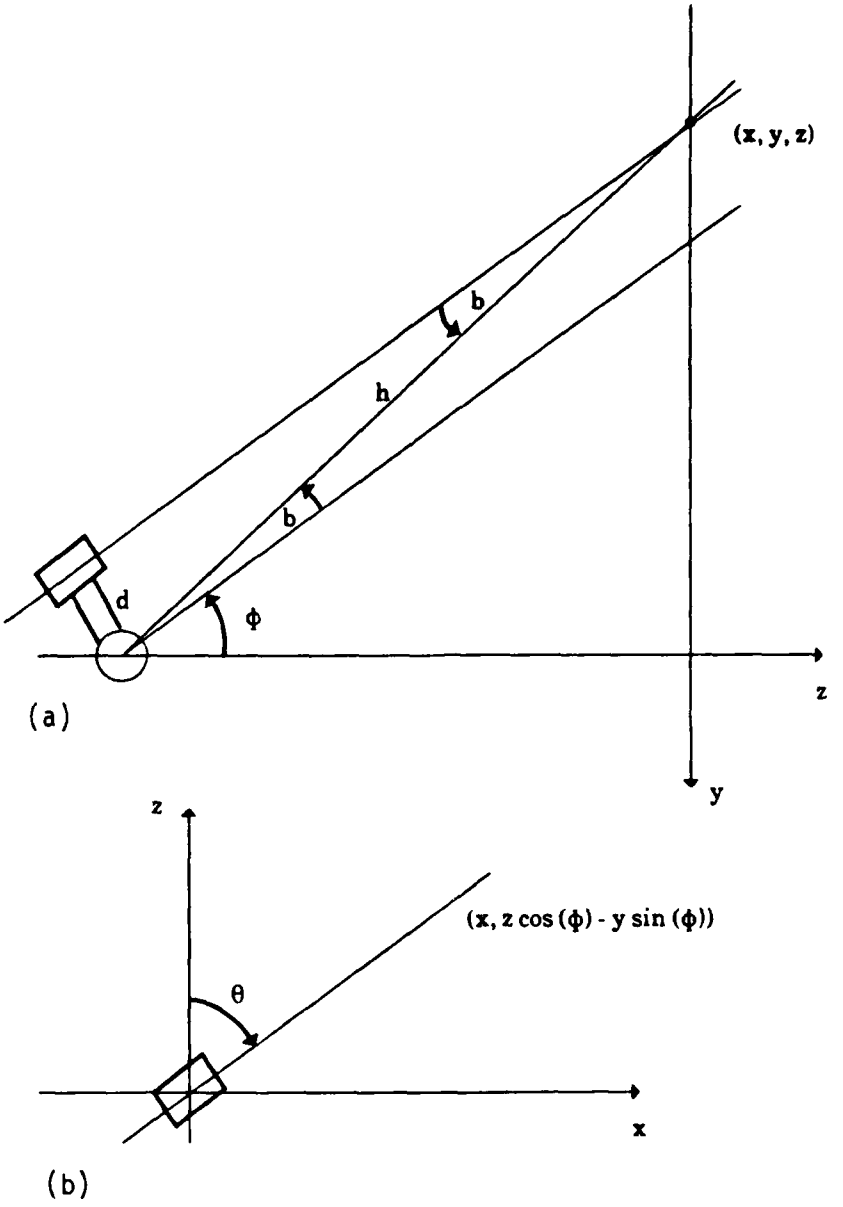


Figure 3: (a) Distances and Angles for computing the ϕ to aim camera at a point $\vec{x}$. (b) Distances to compute θ to aim camera at $\vec{x}$.

11. Inverse Optics: (x,y,z) from Two Images via Pseudoinverse

The complete imaging model is

$$(\hat{x},\hat{y},\hat{z},1)^T = \mathrm{P}(norm\,(\mathrm{C}\vec{x})),$$

where P is the PhysPixel transform that shifts the origin of pixel coordinates to the upper

left corner from the center of the image, *norm*() is the homogeneous vector normalizing operation, C is the imaging matrix given above in the Camera Model section, and T is the inverse of the transform that locates the camera in LAB coordinates, i.e. it is $(FLANGE \cdot CamPos)^{-1}$. First, let

$$(\tilde{x},\tilde{y},\tilde{z},1)^T = P^{-1}(\hat{x},\hat{y},\hat{z},1),$$

(In our system this just amounts to defining the new variables $\tilde{x}=\hat{x}-255, \tilde{y}=\hat{y}-255$). Then from the definition of *norm*(), and letting $\vec{x}=(x,y,z,1)^T$ we have

$$(\tilde{x},\tilde{y}) = \left(\frac{[CT]_0\vec{x}}{[CT]_3\vec{x}}, \frac{[CT]_1\vec{x}}{[CT]_3\vec{x}}\right),$$

where $[CT]_0$ is the first row of CT, etc. Moving the denominators over to the left gives us

$$\tilde{x}[CT]_3\vec{x} = [CT]_0\vec{x}$$

$$\tilde{y}[CT]_3\vec{x} = [CT]_1\vec{x},$$

and multiplying everything out and rearranging gives two linear equations in x,y, and z in terms of the known quantities *f* the effective focal length, *s* the aspect ratio, and T_{ij}.

$$x(\tilde{x}T_{20}-fT_{00})+y(\tilde{x}T_{21}-fT_{01})+z(\tilde{x}T_{22}-fT_{02}) = fT_{03}-\tilde{x}T_{23}$$

$$x(\tilde{y}T_{20}-fsT_{10})+y(\tilde{y}T_{21}-fsT_{11})+z(\tilde{y}T_{22}-fsT_{12}) = fsT_{13}-\tilde{y}T_{23}.$$

Thus knowing the physical locations of two cameras, and knowing the pixel coordinates of the corresponding two images of the same three-dimensional point $\vec{x}$ yields four equations in the three unknowns (x,y,z). They can be solved by a pseudo-inverse method. If X is the matrix of coefficients of (x,y,z) in the above equation and Y is the row matrix of the right hand sides, then the four equations can be written

$$Y=XB$$

if B is the formal column-vector of the variables $(x,y,z)^T$. The values of x,y, and z are obtained simply by computing the pseudo inverse of X:

$$B=(X^TX)^{-1}X^TY.$$

The physical interpretation of this method is made difficult by the fact that the "observables" (the $\hat{x}$ and $\hat{y}$) and the "independent variables" (the T_{ij}) contribute to coefficients of both the B matrix and Y vector. Analysis shows that the effect of noise on this method may be significant, since a one-pixel error in $\hat{x}$ position causes a 20mm depth error at a two meter distance, and a one degree error in azimuth produces a 20mm error in x location. The method has been implemented and integrated into a system that obtains three-dimensional position and verifies it by touching the object with a pointer, and seems to perform as well as the more geometrically intuitive method given in the next section. One potential advantage of the pseudoinverse method is its straightforward extension to more data points.

12. Inverse Optics: (x,y,z) from Two Images via Vectors

Duda and Hart [1973] present a geometrically intuitive method for stereo from two image points. We have implemented it and find it works as well as the pseudoinverse method for two images. The stereo problem is posed using plain 3-vectors, not homogeneous vectors. The following vectors are defined (Fig. 4).

The two cameras have lens centers at $\vec{L}_L$ and $\vec{L}_R$. The vector $\vec{\delta}$ points from $\vec{L}_L$ to $\vec{L}_R$. The 3-D point in the scene is $\vec{x}$. $\vec{x}$ is imaged in cameraL as $\vec{x}_L=(\hat{x}_L\ \hat{y}_L)^t$, and in cameraR it is imaged into $\vec{x}_R$. The vector $\vec{r}_L$ points from the lens center of cameraL through the point $\vec{x}_L$ and to the 3-D point in the scene. A unit vector in the same direction is $\vec{u}_L$. Similarly, cameraR has $\vec{r}_R$ and $\vec{u}_R$ defined.

A temporary world coordinate system is placed at $\vec{L}_L$, thus $\vec{L}_L = \vec{0}$. (Once the 3-D point is estimated in this coordinate system, we will generally convert it to another system such as FLANGE.)

The vectors $\vec{r}_L$ and $\vec{r}_R$ are defined as follows

$$\vec{r}_L=a_L\vec{u}_L,$$

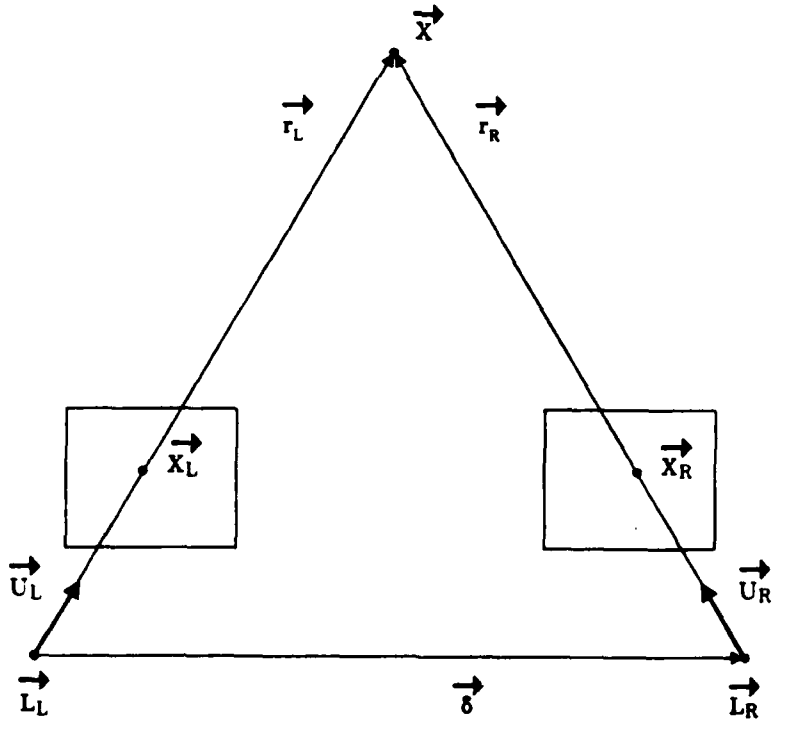


Figure 4. Vectors Used in Two-Vector Stereo Formulation.

$$\vec{r}_R = a_R \vec{u}_R + \vec{\delta}.$$

The approach is to estimate the two scalars a_L and a_R such that the $\vec{r}_L$ and $\vec{r}_R$ vectors are as close together as possible. Note that $\vec{\delta}$ is known and that $\vec{u}_R$ and $\vec{u}_R$ can be computed from $\vec{x}_L$ and $\vec{x}_R$. Then the value of $\vec{x}$ can be computed from a_L and a_R as the point midway between the heads of the $\vec{r}_L$ and $\vec{r}_R$ vectors

$$\vec{x} = [a_L \vec{u}_L + (\vec{\delta} + a_R \vec{u}_R)]/2.$$

The values of a_L and a_R are estimated by minimizing

$$||a_L \vec{u}_L - (\vec{\delta} + a_R \vec{u}_R)||^2.$$

Duda and Hart give the answer to the minimization as

$$a_L = \frac{+\vec{u}_L \cdot \vec{\delta} - (\vec{u}_L \cdot \vec{u}_R)(\vec{u}_R \cdot \vec{\delta})}{1 - (\vec{u}_L \cdot \vec{u}_R)^2},$$

$$a_R = \frac{-\vec{u}_R \cdot \vec{\delta} + (\vec{u}_L \cdot \vec{u}_R)(\vec{u}_L \cdot \vec{\delta})}{1 - (\vec{u}_L \cdot \vec{u}_R)^2}.$$

A few ancilliary equations are

$$\vec{u}_L = \frac{(\hat{x}_L \; \hat{y}_L / s \; f)^T}{||(\hat{x}_L \; \hat{y}_L / s \; f)^T||},$$

$$\vec{u}_R = \frac{(\hat{x}_R \; \hat{y}_R / s \; f)^T}{||(\hat{x}_R \; \hat{y}_R / s \; f)^T||},$$

where s is the pixel aspect ratio. Given the positions of the Left and Right cameras in FLANGE coordinates (call these CamPos transformations CamPosL and CamPosR), $\vec{\delta}$ may be computed as

$$\vec{\delta} = \vec{L}_R = CamPosL^{-1} \, CamPosR \; (0 \; 0 \; 0 \; 1)^T.$$

If the cameras do not have parallel optic axes, then the transformation ($CamPosL^{-1} CamPosR$) must be applied to $\vec{u}_R$ to rotate it with respect to u_L.

13. Lab Constants

These values are taken from the directory /u/brown/robot/include, where there are several files of the form Xconsts.h.

13.1. Head Constants

NECK_OFFSET	(-149.2)	/*pitch axis to tool axis*/
LEFT_OFFSET	(-12.7)	/*tool Z axis to Left camera yaw axis*/
RIGHT_OFFSET	(152.4)	/*tool Z axis to Right camera yaw axis*/
ALT_OFFSET	(-65.1)	/*cam axis to pitch(altitude) axis */
AZ_OFFSET	(34.9)	/*nodal point to yaw(azimuth) axis */

13.2. Camera Constants

```
CAM_F       980.5     /*imaging system effective focal length*/
CAM_S       1.289     /*imaging system pixel aspect ratio y/x*/
```

/* the following constants allow computation of how many pixels to move for a corresponding change in angle, and vice-versa. They are accurate near image center, off by a few pixels in the periphery due to failure of small-angle assumption. The focus distance at which they are computed is the "standard focus distance " of 134cm, by RP and CB on 5 July. The CAM_F above also applies at this distance. */

```
CAM_X_P_D   17.75     /*pixels per degree in x */
CAM_Y_P_D   22.79     /*pixels per degree in y */
```

13.3. Robot Constants

```
INIT_X      650.0     /*init xyzoat when all joint angles 0 */
INIT_Y      190.13
INIT_Z      975.0
INIT_O      90.0
INIT_A      (-90.0)
INIT_T      0.0
```

14. Rigid Transformation Library

Edmond Lee wrote a library for manipulating rigid transforms as an extension to libmatrix, based on an ealier column-vector version by Dave Coombs. It provides basic and efficient implementations of standard data structures and operations. Following are exerpts from its header file.

```
typedef struct matrix *pt_t;                    /* homog point (4x1 col. vector) */
typedef struct matrix *tr_t;                    /* homog transformation (4x4 matrix) */

pt_t pt_zero();                                 /* Return new point 0 0 0 1*/
pt_t pt_rotx(/*pt_t p, double r*/);             /*sideeffects p, rot about X by r */
pt_t pt_roty(/*pt_t p, double r*/);
pt_t pt_rotz(/*pt_t p, double r*/);
pt_t pt_translate(/*pt_t p, double x,y,z*/);
pt_t pt_norm(/*pt_t p*/);                       /* sideeffects p by normalizing it */
pt_t pt_transform(/*pt_t p, tr_t T, pt_t q*/);  /*q = Tp.  returns q*/

tr_t tr_ident();                                /* returns Identity transform */
tr_t tr_rotx(/*tr_t A, double r*/);             /*sideeffects A, rot about X by r*/
tr_t tr_roty(/*tr_t A, double r*/);
tr_t tr_rotz(/*tr_t A, double r*/);
tr_t tr_translate(/*tr_t A, double x,y,z*/);
tr_t tr_invert(/*tr_t A, B*/);                  /* B is A inverted, B returned */
```

```
tr_t tr_transform(/*tr_t A, tr_t B, tr_t C */);        /* C = A*B, C returned */
```

15. Sample Robot Data Structures and Functions

Data structure requirements for applications vary, but the following sort of structures have been successfully used for various vision tasks, and may serve as a template for future (or possibly even standard) robot and head data structures, access functions, and update functions.

```
/*-----------------------------------------------------------------------*/
/* intrinsic camera properties*/

typedef struct timeval *timestamp_t;

typedef struct Camera
{
    int             verbose
    timestamp_t     Ca_Time;            /*time last modified*/
    double          Ca_Focus_Dist;
    double          Ca_Fstop;
    pt_t            Phys_to_Pixel;      /*x and y shifts for phys to pix coords.  */

    tr_t            Ca_Lens;            /*C matrix containing f and sf */
} *Camera_t;

/*-----------------------------------------------------------------------*/
/* Head geometry */

typedef             struct Head_Config  /*uses cnsts in headconsts.h, camconsts.h */
{
    int             verbose;
    timestamp_t     Hd_Time;
    double          Hd_Alt;             /*head  altitude angle (pitch) */

    pt_t            Hd_Nose;            /* offset of end of nose (FLANGE coords) */

/* ---------- Left Camera ----------*/

    double          Hd_AzL;             /*left camera azimuth*/

    pt_t            Hd_CamL_Axis;
/* This vector holds the neck and the camera yaw axis offset, specifying offset from
FLANGE origin to last rigid point in head kinematic chain. Useful intermediate
transform if only eyemovements are happening. */
    tr_t            Hd_CamL_Pos;        /* CamPos: camera's position in FLANGE.*/
    tr_t            Hd_CamL_Inv;        /* Inverse of CamL_Pos */
```

```
        Camera_t        Hd_CamL;/*Left Camera state*/

/* -------- Right Camera is Similar ----------------------------------*/

    double      Hd_AzR;
    pt_t        Hd_CamR_Axis;
    tr_t        Hd_CamR_Pos;
    tr_t        Hd_CamR_Inv;
    Camera_t    Hd_CamR;

} *Head_Config_t;


/*-----------------------------------------------------------------------*/
/* Puma Geometry */

#define PUPDUE 0
#define TYPE 1
#define SIM -1

typedef struct Rob_Config

{
    int             verbose;
    int             Rob_man_sim;r     /* purdue, console, or simulated*/
    int             Rob_ddAlvin;      /*purdue puma device descriptor*/
    timestamp_t     Rob_Time;

    double          Rob_Speed[2];     /*[0] is speed, [1] is mode */
    double          Rob_Jnt[6];       /*Joint angles of Joints 1-6 in order*/
    double          Rob_Location[6];
/*Current TOOL location as X,Y,Z,O,A,T.  Updated as Robot moves.  */

    double          Rob_Tool[6];
/*Current TOOL transform as X,Y,Z,O,A,T. Set by user, same as A7, defines TOOL in
relation to T6, does not move with robot.  */
    tr_t            Rob_FLANGE;       /*transform to move LAB to FLANGE */
    tr_t            Rob_Fl_Inv;       /*inverse of above */

    Head_Config_t   Rob_Head;         /*Head configuration struct*/


}   *Rob_Config_t;

/*------------------Functions ----------------------------*/
/*rob_kine.c */
```

```
extern tr_t             Loc_To_FLANGE ( /* Loc (xyzoat) array */);
/* Returns a CS for the Rob_FLANGE entry in the Config */

extern Rob_Config_t  Rob_Setup(/*simulation type*/);
/*Creates Rob config, also calls head config setup */

extern void             Rob_Set_Tool(/*Loc*/);
/* Set the TOOL CS to be that represented by Loc */

extern void             Rob_Free( /*Rob_Config_t */); /*destroys structure*/

extern void             Rob_Move(/* Rob_Config_t, Rob_Loc */);
/*Depending on mode, moves robot or not. Updates structures.*/

extern void             Rob_Dump(); /*print robot state */


/*head_kine .c */

extern Head_Config_tHead_Setup();
/* Creates the head_config, puts in pre-computable tr_t's */

extern void             Head_Move(/* Head_Config_t, Left_Az, Right_Az, Alt*/);
/*computes  L and R Pos and Cam matrices in FLANGE coords, moves or not depending
on robot mode. */

extern void             Head_Free(/*Head_Config_t */);
extern void             Head_Dump(/*Head_Config_t */);
extern void             Cam_Dump(/*Camera_t */);
```

16. References

Duda, Richard and Peter Hart, *Pattern Classification and Scene Analysis*, John-Wiley & Sons, New York, 1973.

Paul, Richard P., *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge MA 1981.

Tsai, R., "An efficient and accurate camera calibration technique for 3D machine vision", Proc. IEEE-CVPR, 364-374, 1986. (also in *IEEE J. of Robotics and Automation, RA-3* 4, 323-344, Aug. 1987.